Estudio y construcción de una interfaz conversacional mediante Generación Aumentada por Recuperación (RAG) para mejora de extracción de información sobre una enfermedad rara (síndrome de Rett)

Estudio y construcción de una interfaz conversacional mediante Generación Aumentada por Recuperación (RAG) para mejora de extracción de información sobre una enfermedad rara (síndrome de Rett)

índice de figuras

índice de tablas

Un resumen en español

Un resumen en inglés, abstract.

Palabras clave, (como un conjunto, no muy amplio, de palabras significativas en el desarrollo del trabajo.

Palabras clave en inglés, keywords

1 Introducción

Aquí explicamos el objetivo del trabajo, la mejora de los resultados de los modelos generalistas en consultas actuales o muy especializadas mediante la búsqueda previa de documentos, definimos el problema e introducimos brevemente los conceptos principales.

2 Qué es un RAG

Como aparece en el trabajo (Wang, y otros, 2024)

Para entender la necesidad de desarrollar la tecnología RAG, debemos profundizar en el concepto de modelos de lenguaje de gran tamaño (o sus siglas en inglés, LLM). Estos son modelos de inteligencia artificial diseñados para procesar, generar y comprender texto con una notable capacidad. Estos modelos están basados en arquitecturas de redes neuronales profundas, como los *Transformers* (Vaswani, y otros, 2017), que permiten analizar grandes volúmenes de texto y aprender patrones complejos del lenguaje.

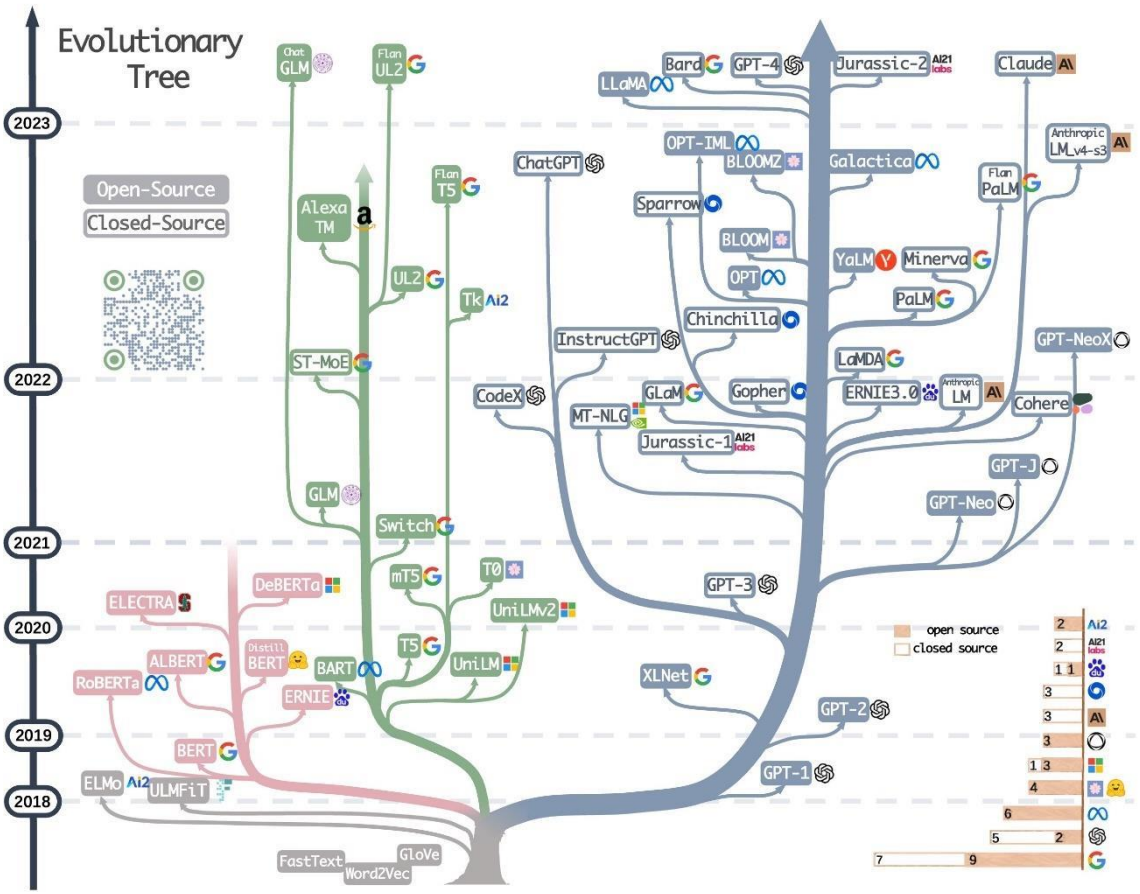

Ilustración 1. Evolución y desarrollo de los modelos de lenguaje a lo largo del tiempo

El proceso de entrenamiento de los LLM se realiza en dos fases principales: el preentrenamiento y el ajuste fino, o en inglés, fine-tuning. Durante el preentrenamiento, el modelo se entrena con enormes cantidades de texto no etiquetado para predecir la siguiente palabra en una secuencia, lo que le permite adquirir conocimientos generales sobre el lenguaje. Posteriormente, el modelo puede ser ajustado mediante fine-tuning con conjuntos de datos específicos y tareas concretas, mejorando su rendimiento en aplicaciones como traducción automática, generación de texto o análisis de conceptos.

El uso de los LLM se ha extendido a numerosos ámbitos, desde la atención al cliente y la creación de contenido, hasta la investigación científica y el desarrollo de software. Su capacidad para generar texto coherente y realizar tareas complejas ha revolucionado la interacción humano-máquina y ha abierto nuevas posibilidades para la automatización y la asistencia en la toma de decisiones.

A pesar de sus ventajas, los modelos preentrenados también presentan limitaciones y desafíos que deben tenerse en cuenta al utilizarlos. Estos pueden estar influenciados por los sesgos inherentes a los conjuntos de datos utilizados en su entrenamiento. Esto puede llevar a que los modelos reproduzcan y amplifiquen sesgos y desigualdades presentes en los datos de entrenamiento original. Además, pueden tener dificultades para adaptarse a dominios o contextos específicos que difieren significativamente de los datos de entrenamiento. Esto puede requerir adaptaciones adicionales o incluso entrenamiento desde cero en algunos casos.

La tecnología RAG, de sus siglas Retrieval Augmented Generation, consiste en la optimización de Modelos de Lenguaje para que puedan dar respuestas más actualizadas y precisas en contextos en los que los modelos típicos no están entrenados. Esta mejora en las respuestas es posible añadiendo contexto a la consulta, y este contexto se crea a partir de una serie de documentos preseleccionados.

Para dar un ejemplo práctico de su funcionamiento, en la siguiente figura se ilustra una aplicación típica de RAG.

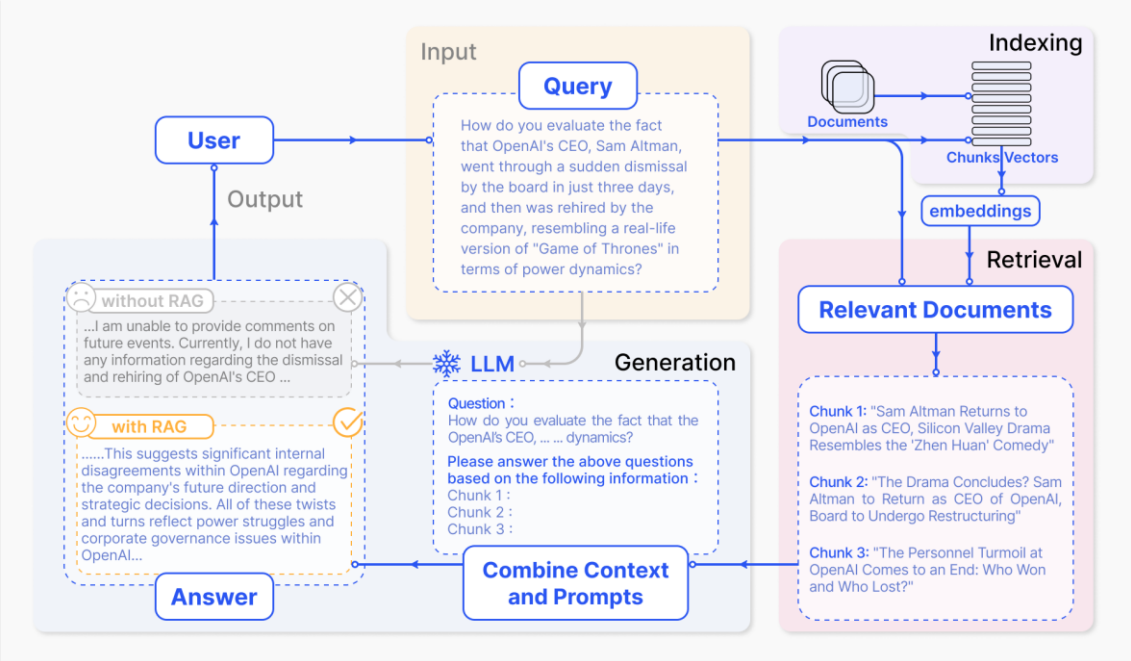

Ilustración 2. Ejemplo típico de RAG (Gao, 2023)

Esta tecnología es especialmente útil para ámbitos demasiado específicos sobre los cuales el modelo tradicional no tiene suficiente información, o para contextos que necesitan información actualizada.

En Ilustración 2, el usuario pregunta a Chatgpt algo relacionado con una noticia reciente. Ya que Chatgpt depende de los datos con los que se entrenó al modelo, no puede dar información actualizada.

La tecnología RAG hace de nodo de interconexión a partir de la información de bases de datos externas. En este caso, recoge artículos de noticias relevantes que tengan que ver con la pregunta del usuario. El modelo se apoya en estos artículos para añadirle contexto a la pregunta, y así dar una respuesta de mayor calidad.

3 Conceptos básicos de un sistema RAG

Para que los documentos en los que queremos que se apoye la aplicación puedan ser usados por el LLM, se pasan a un formato conocido **tokens**, y luego se dividen en **chunks**. Después, estos chunks se pasan a una representación vectorial con un proceso conocido **embedding**, y se guardan en una **base de datos vectorial**.

El chunking es una técnica genial utilizada en la IA generativa para manejar grandes cantidades de datos dividiéndolos en piezas más pequeñas y manejables llamadas **chunks**. Es una técnica crucial que consiste en fragmentar grandes bloques de texto en segmentos más pequeños y fáciles de gestionar. Este proceso es especialmente importante al trabajar con modelos de lenguaje de gran tamaño (LLM) y sistemas RAG, ya que afecta directamente a la relevancia y precisión de los resultados obtenidos por estos modelos.

Sin embargo, una vez decidido el proceso de dividir nuestro corpus, o base de conocimiento del RAG, también es importante decidir cuál será el tamaño de cada chunk. Es por esto por lo que debemos definir una unidad para poder dividir el texto en fragmentos de tamaño similar.

Los modelos de lenguaje usan como unidad básica de texto un concepto conocido como **token**. Esta división se hace por medio de un modelo de tokenización, integrado en los modelos de lenguaje. Cuando un texto se divide en diferentes tokens, la unidad se establece por medio de un algoritmo tokenizador propio del modelo de lenguaje. Cada token se asocia con una palabra, fragmentos de una palabra o caracteres dentro del vocabulario del modelo, y tiene un identificador numérico que le corresponde. Por ejemplo, el carácter " ", es decir, el espacio en blanco, tiene como equivalente el identificador 220.

A continuación, en la Ilustración 3, se muestra un ejemplo de un texto dividido en tokens, usando el algoritmo tokenizador de GPT-4.

El proceso por el cual se decide en cuántos tokens se va a dividir un texto y qué representa cada token en el modelo GPT-4 viene dado por un algoritmo llamado BPE (Byte Pair Encoding). Para dividir el texto, el modelo reconoce las palabras más comunes como vocabulario con el que se entrenó al modelo. Es por esto por lo que palabras simples como "En", "no", "nombre" y similares están representadas por un único token, mientras que otras palabras como "astillero" están divididas en tokens que representan partes de palabras más comunes, como el sufijo "ero".

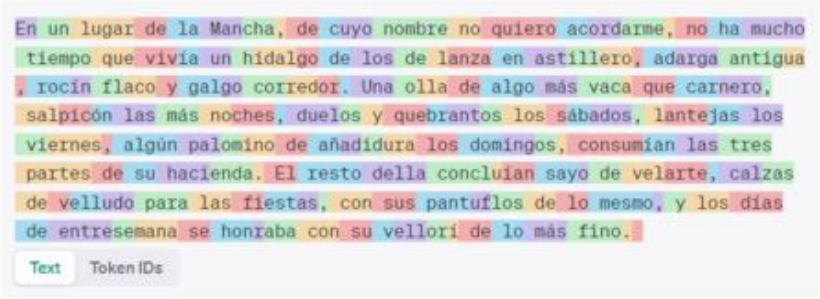

Ilustración 3. División de un texto básico en tokens

En el siguiente fragmento de código, de muestra cómo un simple texto se divide en tokens, mostrándose en orden, el identificador de cada token en el vocabulario del modelo, el número de tokens en el que se ha deconstruido el texto, y el equivalente de cada identificador, (es decir, cada token en formato de lenguaje natural).

```
import tiktoken
import textwrap

texto = "No camines delante de mí, puede que no te siga. No camines detrás de mí, puede que no te guíe. Camina junto a mí y sé mi amigo.  ~Albert Camus"

tokenizer = tiktoken.get encoding("cl100k base")  #Usado en GPT-4 y en el modelo de embedding ada-002 de OpenAI
```

```
tokens = tokenizer.encode(texto)
print("Tokens:")
print(tokens)
print("Cantidad de tokens:", len(tokens))
print("Tokens decodificados:")
print(str([tokenizer.decode([t]) for t in tokens]))
```

```
Tokens:
[2822, 6730, 1572, 1624, 5048, 409, 77426, 11, 21329, 1744, 912,
1028, 8531, 64, 13, 2360, 6730, 1572, 35453, 7206, 409, 77426, 11,
21329, 1744, 912, 1028, 1709, 2483, 68, 13, 8215, 2259, 63088, 264,
77426, 379, 19266, 9686, 71311, 13, 220, 4056, 67722, 8215, 355]

Cantidad de tokens: 46

Tokens decodificados:
['No', ' cam', 'ines', ' del', 'ante', ' de', ' mí', ',', ' puede',
' que', ' no', ' te', ' sig', 'a', '.', ' No', ' cam', 'ines', '
detr', 'ás', ' de', ' mí', ',', ' puede', ' que', ' no', ' te', '
gu', 'í', 'e', '.', ' Cam', 'ina', ' junto', ' a', ' mí', ' y', '
sé', ' mi', ' amigo', '.', ' ', ' ~', 'Albert', ' Cam', 'us']
```

Luego, se muestra un ejemplo en el que un texto se divide en chunks, con tamaño de 20 tokens y una solapación (overlap) de 5 tokens. Este último concepto consiste en que cada chunk, con excepción del primero, empieza un poco antes de que acabe el anterior, para no perder contexto. En el apartado **Descomposición en chunks** se explica en mayor detalle este concepto de solapación.

```
from langchain.text_splitter import TokenTextSplitter

texto = "Caminante, son tus huellas el camino y nada más;
Caminante, no hay camino, se hace camino al andar. Al andar se hace
el camino, y al volver la vista atrás se ve la senda que nunca se
ha de volver a pisar. Caminante no hay camino sino estelas en la
mar."

#Se establece el tamaño de cada chunk
splitter = TokenTextSplitter(chunk_size=20, chunk_overlap=5)

chunks = splitter.split_text(texto)

for i, chunk in enumerate(chunks):
    print(f"Chunk {i + 1}: {chunk}")
```

```
Chunk 1: Caminante, son tus huellas el camino y nada más; C

Chunk 2: ada más; Caminante, no hay camino, se hace camino al and

Chunk 3: ace camino al andar. Al andar se hace el camino, y al vol

Chunk 4: ino, y al volver la vista atrás se ve la senda que nunca

Chunk 5:  senda que nunca se ha de volver a pisar. Caminante no hay

Chunk 6:  Caminante no hay camino sino estelas en la mar.
```

Una vez establecida la división de nuestro texto en un formato más manejable para el modelo de lenguaje, es necesario convertirlo a un formato en el que sea más facil recuperar la información. Para nosotros, resulta fácil entender conceptos abstractos como la justicia o la belleza, pero los computadores necesitan un formato matemático en el que poder procesar esta información, por esto se utilizan un tipo específico de vectores, los **embeddings.**

El concepto de *embedding* consiste en una técnica de representación de información en el que, para cada concepto, se evalúan todas sus características de una forma numérica, mediante el uso de un algoritmo y se guarda como un vector de distintos números. Luego cada posición de cada vector se trata como dimensiones en un espacio, en el que cada vector o concepto, significa un punto dentro de este espacio multidimensional.

Una vez se han transformado los tokens en *embeddings*, se genera una asociación entre todas las listas de vectores, para poder darle una representación vectorial al *chunk* por completo. En el modelo de *embedding* "ada-002" de OpenAI, se calcula el promedio de todas las representaciones vectoriales de cada token para generar la representación vectorial de su *chunk*.

Lo siguiente sería analizar dónde se guarda esta información, y la respuesta está en las bases de datos vectoriales. En lugar de la clásica base de datos relacional, con tuplas y columnas, cada elemento se representa con un vector, que en definitiva significa un punto dentro de un complejo espacio euclídeo de amplias dimensiones (el modelo de *embeddings* de OpenAI "ada-002" utiliza 1536 dimensiones distintas para calcular la posición de cada vector). Algunos ejemplos comerciales de instancias de esta tecnología son FAISS (*Facebook AI Similarity Search*), Milvus y Pinecone.

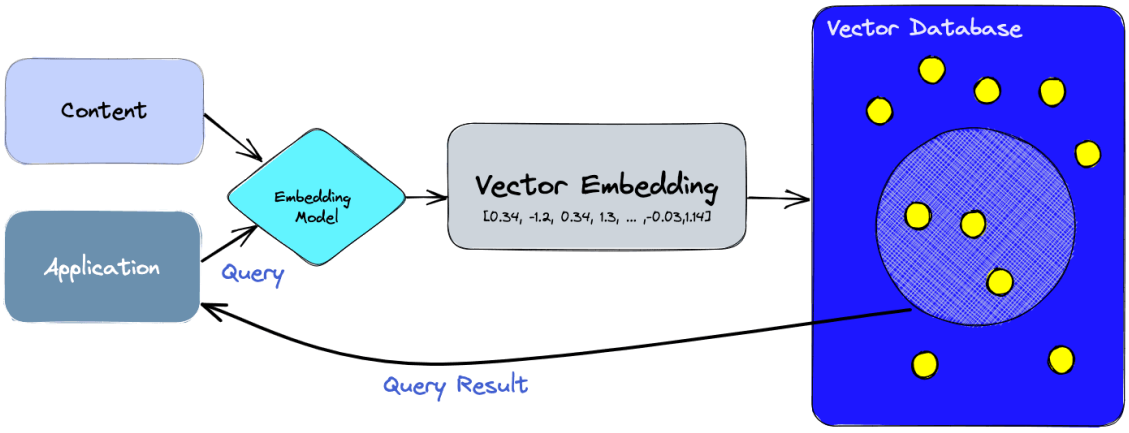

Ilustración 4. Demostración visual de una base de datos vectorial

la Ilustración 4

Esto es crucial para poder establecer búsquedas de similitud en la fase de recuperación. Al tratar con vectores que significan puntos en el espacio, se puede establecer una similitud entre ellos. Cuanto más cerca estén en el espacio, más parecidos serán los conceptos. Para medir la distancia entre ellos, se puede calcular la distancia euclídea o el coseno del ángulo que forman.anteriormente

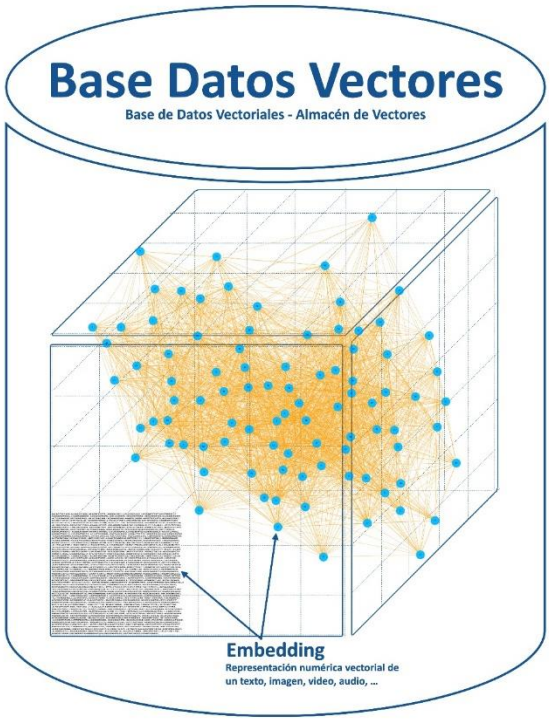

Ilustración 5. Búsqueda por similitud en base de datos vectorial

4 Estructura del desarrollo de un sistema RAG

La investigación de modelos RAG está en constante desarrollo, y la dividimos en 3 fases. RAG ingenuo, RAG avanzado y RAG modular, como se puede ver en la siguiente figura.

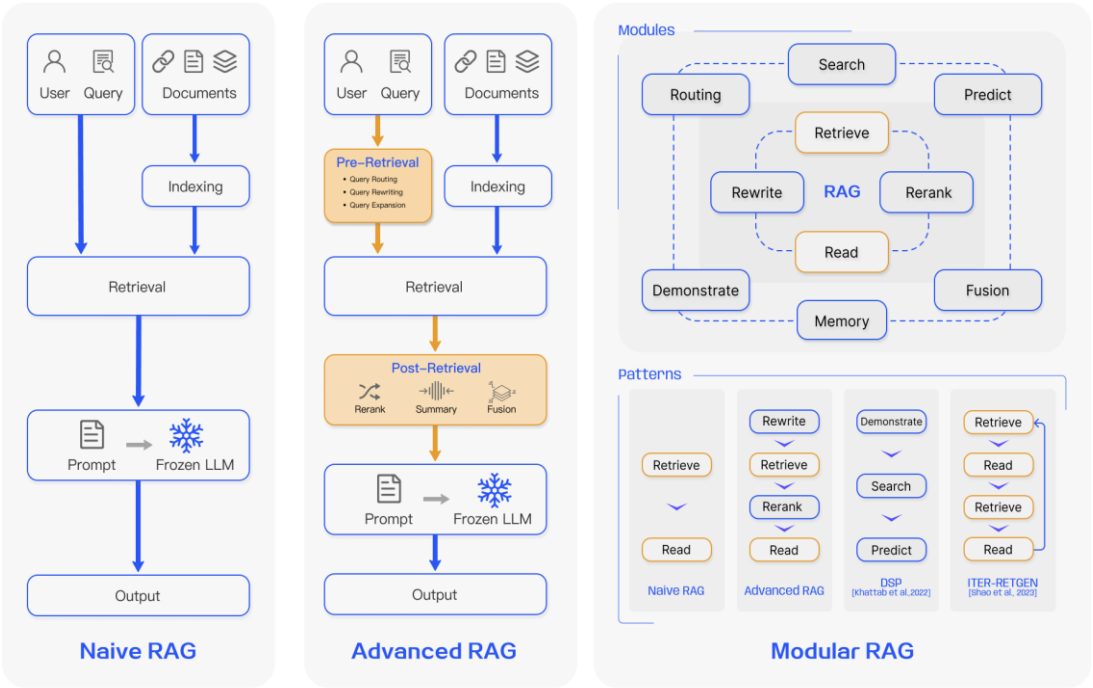

Ilustración 6. Representación en diagramas de las fases de un RAG (Gao, 2023)

Aunque los modelos RAG son muy útiles para enriquecer los LLM, hay que tomar conciencia de las limitaciones que conllevan.

4.1 RAG Ingenuo (Naive RAG):

Representa la metodología más primitiva dentro del paradigma de investigación del RAG. El RAG ingenuo sigue un proceso que incluye 3 fases: indexación, recuperación y generación.

La indexación comienza con la extracción y refinación de datos en distintos formatos como PDF, HTML, Markdown, Word, etc.., que se convierten a texto plano. Luego, como hemos visto en Conceptos básicos de un sistema RAG, se convierte a un sistema de representación numérica y se guarda en una base de datos vectorial para poder efectuar búsquedas por similitud.

En la fase de recuperación, se recoge la consulta del usuario, y el sistema RAG ejecuta el mismo proceso que en la fase de indexación para pasar la consulta a una representación vectorial. Tras ello, calcula la similitud entre el vector de la consulta y el vector de chunks almacenado en la base de conocimiento del LLM. El sistema prioriza y recupera los chunks que poseen mayor similitud a la consulta. Estos chunks se usan como contexto expandido en el prompt.

En la generación, la consulta y los documentos seleccionados se sintetizan en un prompt coherente, y el LLM deberá producir una respuesta acorde. La estrategia del modelo a la hora de responder podrá variar dependiendo según el criterio específico de la tarea, pudiendo ceñirse estrictamente a los documentos proporcionados o sacar sus propias conclusiones. Si llega a haber un diálogo, cualquier conversación en el historial se puede integrar en el prompt, habilitando al modelo a seguir una conversación de forma efectiva.
Sin embargo, el RAG ingenuo posee algunas desventajas.

- Dificultad para recuperar: en la fase de recuperación, a menudo surgen problemas relacionados con la precisión, seleccionando chunks irrelevantes, y falta de información crucial.

- Problemas para generar: a la hora de generar respuestas, el modelo puede sufrir alucinaciones, y producir contenido que no está basado en el contexto aportado. Esta fase también puede proporcionar respuestas irrelevantes o sesgadas, lo cual daña la calidad de las respuestas.

-*Desafíos en la Aumentación: Puede resultar complicado evaluar la relevancia y la importancia de los diferentes fragmentos de información, así como mantener la coherencia en el estilo y el tono del contenido generado.*

Cuando se abordan temas complejos, una única recuperación basada en la consulta original suele ser insuficiente para obtener el contexto necesario. De igual forma, si la consulta requiere una respuesta simple, y el modelo recupera un contexto extenso, es posible que se presenten alucinaciones a la hora de generar la respuesta. A esto se suma el riesgo de que los modelos de generación dependan en exceso de la información recuperada, limitándose a repetir el contenido obtenido sin aportar análisis, ideas nuevas o información sintetizada.

4.2 RAG Avanzado (Advanced RAG)

El RAG avanzado introduce una serie de mejoras para superar las limitaciones del RAG ingenuo. Se centra en mejorar la calidad de la recuperación, mediante estrategias pre-recuperación y

post-recuperación. Para sobrellevar los problemas de indexación, esta versión refina sus técnicas de indexación mediante el uso de una ventana deslizante, segmentación refinada y la incorporación de metadatos. Además, incorpora varios métodos de optimización para el proceso de recuperación.

- Pre-recuperación: en este proceso, el objetivo principal es optimizar *la consulta original, para que el modelo pueda comprender mejor lo que el usuario requiere y para que la indexación localice más fácilmente los recursos a buscar.*

Para optimizar la consulta del usuario, se suelen usar métodos como división en subconsultas o reescritura de consulta.

- Post-recuperación: Una vez que tenemos el contexto, es crucial integrarlo de forma efectiva con la consulta. Los principales métodos incluyen la reordenación de chunks y contracción de texto.

Reordenar la información recuperada para encontrar el contexto más relevante de cara al prompt es un paso clave. Este proceso ya ha sido integrado en diversos frameworks como LlamaIndex, Langchain y HayStack.

Otorgar todos los documentos relevantes directamente al LLM, pueden desencadenar en una sobrecarga de información, menospreciando los detalles más importantes. Para combatir esto, los métodos post-recuperación se centran en seleccionar la información esencial, reduciendo el tamaño del contexto que se envía.

4.3 RAG Modular (Modular RAG)

La arquitectura del RAG incorpora varias estrategias para mejorar sus componentes, como añadir un módulo de búsqueda de similitudes y redefinir la recuperación mediante fine-tuning.

Estas innovaciones tratan de incluir distintos módulos para mejorar cualidades específicas del sistema RAG. La transición a un enfoque modular se está volviendo predominante, puesto que, debido a su estructura secuencial, permite un entrenamiento individualizado en sus componentes. A continuación, se enumeran algunos de los diferentes módulos que se pueden integrar en un sistema RAG.

1) **Nuevos módulos:**

Módulo de búsqueda: se adapta para escenarios específicos, habilitando búsquedas directas a través de varias fuentes de datos como motores de búsqueda, bases de datos y gráficos de conocimiento.

Módulo de fusión: resuelve los problemas de búsquedas tradicionales mediante una estrategia multi-consulta que divida la consulta del usuario en distintas perspectivas, usando búsquedas vectoriales de forma paralela y reordenación inteligente para poder interpretar nuevos conocimientos.

Módulo de memoria: aprovecha la memoria del LLM para guiar la recuperación, creando un conjunto de memoria ilimitado que alinea el texto *de una forma más cercana a* la distribución de datos mediante un proceso iterativo de auto-mejora.

Módulo de enrutamiento: navega por diferentes fuentes de datos, mezclando diferentes flujos de información.

Módulo de predicción: intenta reducir la redundancia y ruido generando contexto directamente desde el LLM, asegurando la precisión a la hora de construir una respuesta.

Módulo adaptador de tareas: conduce al RAG a varias tareas secuenciales, automatizando la recuperación de prompts. Este enfoque no solo simplifica el proceso de recuperación, sino que también mejora la calidad y relevancia de la información recibida, alcanzando un mayor abanico de tareas y solicitudes que poder completar con eficiencia y exactitud.

2) **Nuevos Patrones:**

Los RAG modulares ofrecen una flexibilidad destacable permitiendo la sustitución de módulos o reconfiguración para objetivos específicos. Esto va más allá de las estructuras fijas de las RAG ingenuo y avanzado, que se caracterizan por un mecanismo simple de "recuperación y lectura".

Además, el RAG modular expande dicha flexibilidad integrando nuevos módulos y ajustando la interacción entre los ya existentes, mejorando sus aplicaciones en distintas tareas.

Algunas innovaciones como *la reescritura o clasificación de consulta* se aprovechan de las capacidades del LLM para refinar consultas de recuperación mediante un módulo de reescritura y un mecanismo de realimentación para el LLM, mejorando el rendimiento.

~~La arquitectura flexible del RAG modular muestra los beneficios de la recuperación adaptable mediante técnicas como FLARE y Self-RAG. Este enfoque transciende la recuperación estática del RAG evaluando la necesidad de la recuperación en distintos escenarios~~. Otro beneficio de la arquitectura flexible es que el sistema RAG puede fácilmente integrarse con otras tecnologías, como fine-tuning o aprendizaje reforzado.

5 RAG vs Fine-tuning

El crecimiento de los LLM ha atraído una atención considerable. Entre los métodos de optimización de modelos de lenguaje, el RAG a menudo se compara con Fine-tuning, e ingeniería de prompt.

La ingeniería de prompt consiste en optimizar las consultas al modelo para guiarle a dar una mejor respuesta.

El fine-tuning consiste en entrenar al modelo original con datos y preguntas relacionadas al contexto pertinente, mientras que en el RAG, estos nuevos datos simplemente se añaden como contexto en la consulta, no forman parte de su base de conocimiento.

Cada método tiene características distintas como se ilustra en Ilustración 2.

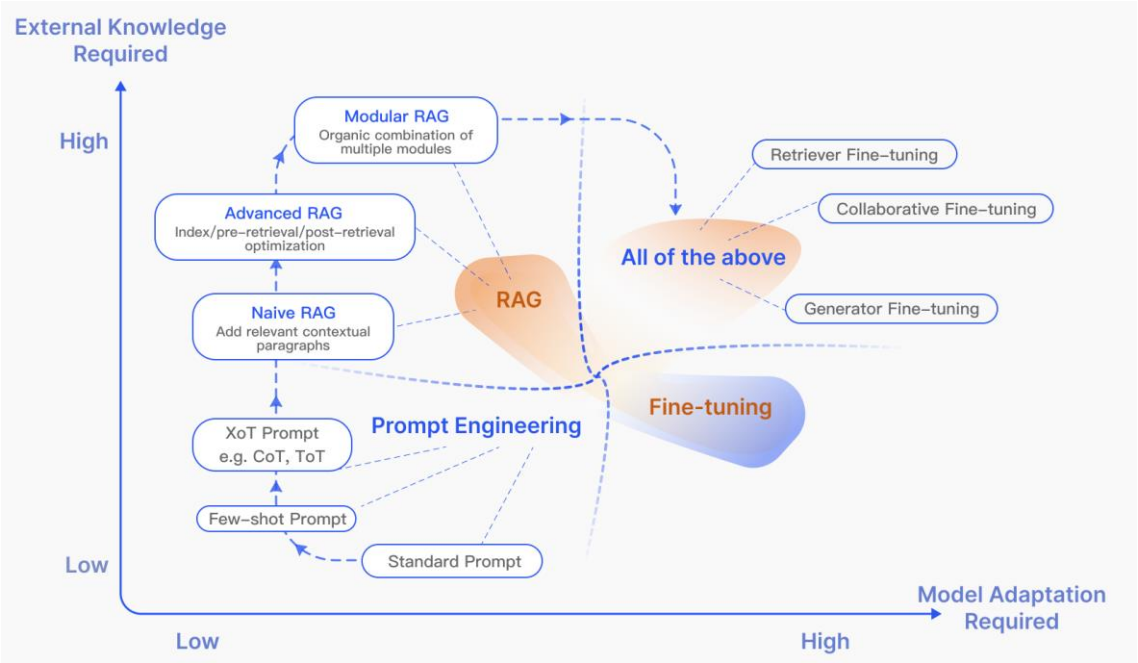

Ilustración 7. RAG comparado con fine-tuning e ingeniería de prompt (Gao, 2023)

El RAG destaca en entornos dinámicos ofreciendo conocimiento actualizado y un uso efectivo de fuentes de conocimiento externas con un amplio grado de interpretabilidad. Sin embargo, conlleva una mayor latencia y consideraciones éticas a la hora de tomar los datos. Por otra parte, el Fine-tuning es más estático, necesitando reentrenamiento para actualizaciones, pero habilitando un mayor grado de configuración del comportamiento y estilo del modelo. Requiere más recursos computacionales para la preparación del conjunto de datos y entrenamiento y, aunque puede reducir el número de alucinaciones, puede tener problemas a la hora de lidiar con datos que no conoce.

~~En múltiples evaluaciones de rendimiento en varias tareas complicadas de temas diferentes~~ realizadas en (Gao, 2023)~~, se~~ concluye ~~que, aunque el fine-tuning sin supervisión muestra alguna mejora, el RAG siempre le supera, tanto para conocimiento existente que ya se encontraba en el entrenamiento tanto como para nuevo conocimiento. Además, también se~~ concluye ~~que a los LLM les cuesta encontrar nueva información contrastada mediante FT sin supervisión~~. (quitar párrafo?)

La elección entre RAG y FT depende de las necesidades específicas de la dinámica de los datos, y las capacidades computacionales *del entorno* de la aplicación. RAG y FT no son mutuamente excluyentes y se pueden complementar entre sí, mejorando las capacidades del modelo en distintos niveles. En determinados casos, su uso combinado puede dar como resultado un rendimiento óptimo.

6 Estructura de un RAG

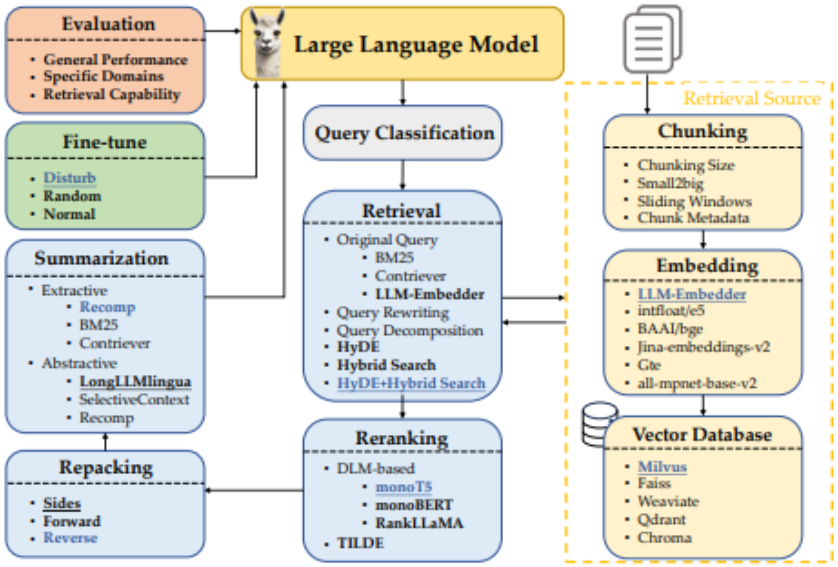

Ilustración 8. Diagrama de componentes de un sistema RAG (Wang, y otros, 2024)

Las técnicas de generación aumentada por recuperación (RAG) han demostrado ser efectivas para integrar información actualizada, reducir alucinaciones y mejorar la calidad de las respuestas en dominios especializados. Sin embargo, estas técnicas a menudo enfrentan desafíos en su implementación y tiempos de respuesta.

En (Wang, y otros, 2024), se analizan los enfoques RAG existentes y sus combinaciones para identificar prácticas óptimas que equilibren rendimiento y eficiencia (ver Ilustración 8. Diagrama de componentes de un sistema RAG). Se proponen estrategias para la implementación de RAG, destacando que las técnicas de recuperación multimodal pueden mejorar significativamente las capacidades de respuesta a preguntas sobre entradas visuales y acelerar la generación de contenido multimodal.

Además, concluyen que los RAG permiten aplicaciones rápidas en organizaciones específicas sin necesidad de actualizaciones en los parámetros del modelo, siempre que se proporcionen documentos relacionados con la consulta.

En esta sección, entraremos en detalle para describir los componentes que forman el flujo de trabajo de un RAG. Para cada módulo, revisaremos cada enfoque y los mejores métodos para cada componente, según (Wang, y otros, 2024).

7 Clasificación de consulta

No todas las consultas requieren usar el RAG por la naturaleza del LLM. Mientras que el RAG puede mejorar la calidad de la información devuelta y reducir el riesgo de alucinaciones, un uso frecuente de la recuperación aumentada puede resultar en un aumento del tiempo de respuesta. Por tanto, empezamos clasificando las consultas para comprobar la necesidad de recuperación. Si no lo requieren, simplemente se usará el LLM por defecto.

La recuperación se necesita cuando el conocimiento requerido va más allá del que posee el modelo. Sin embargo, la necesidad de recuperación varía según la tarea. Por ejemplo, un LLM al

que se pide una traducción, podrá responder sin problemas, pero si se le pregunta una duda técnica sobre una tecnología que no conoce, necesitará recuperar información.

Por tanto, se propone clasificar las consultas por tareas. La división se ha establecido en 15 tipos diferentes de tareas. Aquellas tareas que se basen en información otorgada por el usuario, serán marcadas como "suficiente", si no es así, serán marcadas como "Insuficiente" y es posible que se necesite la recuperación. Se ha entrenado un clasificador para automatizar este proceso.

8 Descomposición en chunks

La división de documentos en segmentos más pequeños es crucial para mejorar la precisión de la recuperación y evitar problemas de longitud de respuesta en los LLM. Este proceso se puede hacer a distintos niveles de tamaño:

- **Nivel Token**: es simple pero puede llegar a dividir frases, afectando a la calidad de recuperación.
- **Nivel Semántico**: usa el LLM para determinar los puntos de división, manteniendo el contexto pero más caro computacionalmente.
- **Nivel frase**: equilibrado entre mantener la semántica del texto y ser eficiente.

En este estudio, usaremos la división por frases. Entraremos en más detalle en función de 4 dimensiones.

- ## Tamaño de los chunks:

  El tamaño del chunk tiene un gran impacto en el rendimiento. Si el chunk es más grande, provee mayor contexto, mejorando la comprensión del modelo, pero aumentando el tiempo de respuesta. Lo contrario ocurre a medida que el chunk es más pequeño.

  Para obtener el tamaño óptimo, se usan varias métricas como fidelidad o relevancia. La fidelidad comprueba que la respuesta sea una alucinación o que se corresponda con el texto recuperado. La relevancia comprueba que el texto recuperado y las respuestas están relacionadas con las consultas.

  En las pruebas, se ha usado una superposición de chunks de 20 tokens. El tamaño que resulta en mayor fidelidad es **512** tokens, mientras que el tamaño con mayor relevancia es **256**.

- ## Técnicas de descomposición de chunks:

  Cuando los textos se dividen en chunks, es posible que se pierda información relevante al hacer el corte entre un chunk y otro, por ejemplo, si el corte se efectúa justo en la mitad de la frase.

  Existen técnicas avanzadas como small-to-big y sliding-window para mejorar la calidad de recuperación organizando relaciones en los bloques de chunks. ~~Los bloques pequeños se usan para emparejarlos con las consultas, y lo que se devuelve son bloques más grandes que incluyen los pequeños con información contextual.~~

  Small-to-big consiste en unir chunks que son demasiado pequeños en chunks más grandes para poder para poder dar contexto más relevante. Sliding-window consiste

en de dejar una solapación entre chunks, de forma que cada chunk empiece ocupando parte del final del chunk anterior.

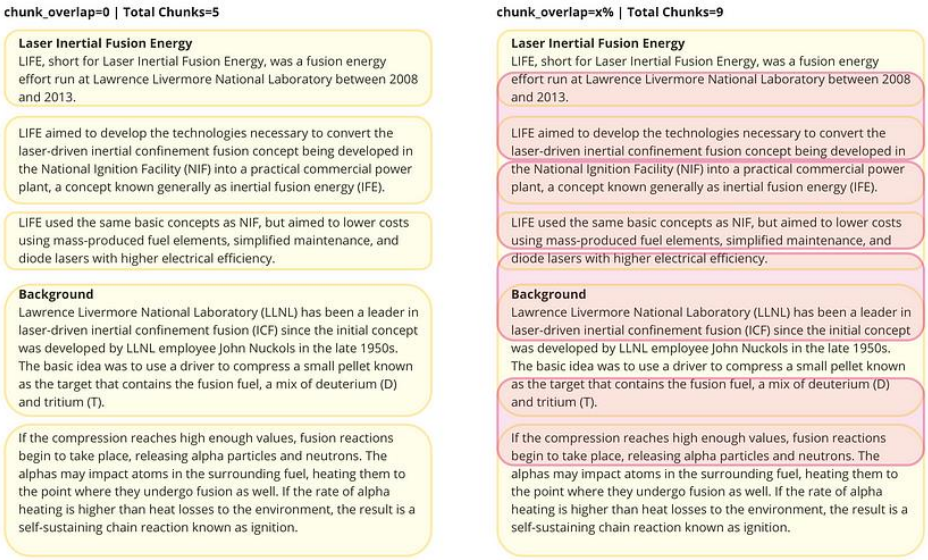

Ilustración 9.. Demostración gráfica de sliding window o superposición de chunks

Para demonstrar la efectividad de estas técnicas avanzadas, se han realizado una serie de pruebas. En estas pruebas, el tamaño de chunk más pequeño es de 175 tokens, y el más grande es de 512, y la superposición es de 20 tokens. Dentro de las técnicas, **sliding-window** es el más efectivo tanto para mantener la relevancia como la fidelidad.

- **Selección de modelo embedding:**

Elegir el modelo de embedding correcto es esencial para mantener la semántica de las consultas y los chunks. Se ha usado el modelo de FlagEmbedding para evaluar los resultados. El que ha obtenido mejores resultados es el modelo **LLM-Embedder**.

- **Añadir metadatos:**

Se puede mejorar la recuperación añadiendo información adicional a los chunks, como títulos, palabras clave y preguntas hipotéticas, para que el LLM entienda mejor la información recuperada.

9 Base de datos de vectores

Las bases de datos de vectores guardan los vectores de embedding con sus metadatos asociados, permitiendo la recuperación eficiente de documentos relevantes mediante varios métodos como ANN (aproximación al vecino más cercano).

Para seleccionar una base de datos adecuada a la investigación, se han elegido 4 criterios: múltiples tipos de índice, soporte de vectores de escala millonaria, búsqueda híbrida y capacidad de ser guardado en la nube. Si tiene múltiples tipos de índice, se asegura la flexibilidad para optimizar búsquedas basadas en distintos tipos de datos. El soporte de vectores de escala es

importante si se quieren manejar conjuntos de datos muy grandes. La búsqueda híbrida combina búsqueda de vectores con búsqueda tradicional de palabras claves.

En la investigación realizada, **Milvus** es el único que cumple todos los requisitos.

10 Métodos de recuperación

Dada una consulta de un usuario, el modelo de recuperación selecciona, los documentos *n*-primeros con mayor relevancia dentro de una base de conocimiento basada en la similitud de la consulta con los documentos. Después, el modelo de generación usa estos documentos para formular una respuesta apropiada a la consulta. Sin embargo, las consultas por defecto a menudo no devuelven el resultado esperado debido a una expresión poco precisa y una falta de información por parte del LLM. Para resolver estos problemas, hemos evaluado 3 métodos para transformar las consultas:

- **Reescritura de consultas:** mejora las consultas para que se acerquen más a los documentos relevantes.
- **Descomposición de consultas:** Este enfoque implica recuperar documentos según unas preguntas derivadas de la consulta original, lo cual resulta más complejo.
- **Generación de pseudo-documentos:** Este enfoque genera un documento hipotético basado en la consulta del usuario y usa un embedding de respuestas hipotéticas para recuperar documentos similares. Una implementación destacable es **HyDE**.

Algunos estudios recientes indican que combinando una búsqueda basada en el léxico con una basada en vectores mejora considerablemente el rendimiento.

11 Métodos de reorganización de chunks

Después de la recuperación inicial, se ejecuta una fase de reorganización para mejorar la relevancia de los documentos recuperados, asegurando que la mejor información se encuentra la primera de la lista. Esta fase usa métodos más precisos e intensivos para funcionar eficazmente.

Consideramos dos enfoques en nuestro módulo de reorganización. **Reorganización DLM**, que emplea clasificación y **Reorganización TILDE**, que se centra en el objetivo de la consulta. Estos enfoques priorizan el rendimiento y la eficiencia, respectivamente.

- **DLM:** Este método aprovecha modelos de lenguaje profundos (DLM) para la reorganización. A estos modelos se les ha hecho un fine-tuning para clasificar la relevancia de los documentos con respecto a la consulta según un criterio de verdadero o falso.
- **TILDE:** Este calcula la probabilidad de cada término de la consulta de manera independiente prediciendo las probabilidades de los tokens a lo largo del vocabulario del modelo. Los documentos se puntúan sumando las probabilidades logarítmicas previamente calculadas de los tokens de la consulta, lo que permite un reordenamiento rápido durante la inferencia.

Una vez realizadas las pruebas, se descubre que RankLLaMa es el que devuelve mejor renidmiento, mientras que TILDEv2 es el ideal para un proceso rápido con una colección fija.

12 Reensamblado de documentos

El rendimiento de subprocesos, como la generación de respuesta del LLM, puede estar afectado por el orden en el que se suministran los documentos. Para solucionar esto, se incorpora un módulo de reensamblado en el flujo de trabajo después de reorganizar, que involucra tres métodos, “directo”, “inverso” y “lateral”. El método directo los ordena de forma descendente, según su puntuación de relevancia en la fase de reorganización, mientras que el método inverso los ordena de forma ascendente. Hay estudios que demuestran que la mayor eficiencia se encuentra al posicionar los documentos al principio o al final de la lista, resultando en el método lateral.

13 Síntesis

Los resultados de la recuperación pueden contener información redundante o innecesaria, lo que impide al modelo de lenguaje de generar respuestas útiles. Además, los prompts largos pueden ralentizar el proceso, por ello, son cruciales los métodos para sintetizar los documentos recuperados en el flujo de trabajo del RAG.

Las tareas de síntesis pueden ser extractivas o abstractivas. Los métodos extractivos dividen el texto en frases, y luego los puntúan y ordenan según su importancia. Los abstractivos sintetizan información de múltiples documentos para reorganizarla y generar un resumen coherente. Estas tareas pueden ser basadas en consultas o no. En este estudio, ya que el RAG devuelve información relativa a las consultas, nos centramos exclusivamente en métodos basados en consultas.

- **Recomp:** tiene sintetizadores extractivos y abstractivos. El sintetizador extractivo selecciona frases útiles, mientras que el sintetizador abstractivo recoge información de varios documentos.
- **LongLLMLingua:** mejora LLMLingua centrándose en información clave relativa a la consulta.
- **Contexto selectivo:** mejora la eficiencia del LLM identificando y retirando información redundante en el contexto de entrada.

Se recomienda **Recomp** por su destacable rendimiento.

14 Fine-Tuning generador

En esta sección nos centramos en hacer fine-tuning al generador mientras dejamos el fine-tuning del recuperador para otro momento. Se busca investigar el impacto del fine-tuning, sobre todo la influencia de contexto relevante o no, en el rendimiento del generador.

Las pruebas sugieren que mezclar contextos aleatorios con contextos relevantes durante el entrenamiento puede mejorar la robustez del generador a información irrelevante mientras se asegura el uso efectivo de contexto relevante. Por ello, se identifica que la práctica de suministrar una mezcla de documentos relevantes con otros aleatorios durante el entrenamiento, es el mejor enfoque.

15 Evaluación

Hemos realizado una serie de pruebas para evaluar el rendimiento de los sistemas RAG. Estas pruebas son: **Razonamiento de sentido común, Comprobación de hechos, Preguntas y respuestas de dominio abierto, Preguntas y respuestas de multi-salto, y Preguntas y respuestas médicas.**

La métrica usada para la evaluación de Razonamiento de sentido común, Comprobación de hechos y Preguntas y respuestas médicas ha sido la precisión, para el resto, se han usado la puntuación F1 a nivel de token y puntuación Exact Match. La puntuación final del RAG fue calculada mediante el promedio de las 5 cualidades mencionadas anteriormente.

Resultados y análisis

Basándonos en los resultados de las pruebas, los aspectos más destacables son:

- **Módulo de clasificación de consultas:** Este módulo se referencia y contribuye tanto a la efectividad como a la eficiencia, dando lugar a una mejoría en la puntuación general de 0,428 a 0,443, y una reducción en latencia de 16,41 a 11,58 segundos por consulta.
- **Módulo de recuperación:** Mientras que el método HyDE híbrido obtuvo la mejor puntuación RAG de 0,58, también requiere un alto coste computacional de 11,71 segundos por consulta. Además, los métodos híbridos u originales son los recomendados, puesto que reducen el tiempo de respuesta manteniendo una eficacia parecida.
- **Módulo de reordenación o reranking:** Si no existe este módulo, se nota una caída en la eficacia de la aplicación, señalando su importancia. MonoT5, acumuló la puntuación más alta, demostrando su eficacia a la hora de aumentar la relevancia de los documentos recuperados.
- **Módulo de reempaquetado o repacking:** La configuración inversa demostró un rendimiento superior. Esto demuestra que posicionar contenido relevante cerca de la consulta aumenta la calidad de las respuestas.
- **Módulo de resumen o sumarización:** Recomp fue el que demostró mejor rendimiento, aunque se pueden conseguir parecidos con menor latencia eliminando este módulo.

Los resultados de los experimentos demuestran que cada módulo contribuye de forma única al rendimiento del sistema RAG al completo. El módulo de clasificación mejora la precisión y reduce la latencia, mientras que los de recuperación y reordenación mejoran la capacidad del sistema para responder una mayor variedad de preguntas. Los de reempaquetado y sumarización afinan la salida, asegurando una mayor calidad de respuestas.

16 Un breve ejemplo. Tutorial explicado con un notebook comentado

Aquí pondría el ejemplo de FAISS con síndrome de Rett y pocos documentos, explicado paso a paso.genéricos cuando

El primer paso dentro del tutorial sería establecer las claves de API de los servicios que vayamos a usar (en este caso usamos la clave de OpenAI) y la instalación de dependencias.

```
!pip install langchain-community unstructured pypdf tiktoken faiss-cpu openai -q
```

Lo siguiente sería declarar el corpus que usará nuestro sistema RAG. En este caso, está compuesto por 11 artículos médicos descargados desde el portal PubMed en formato PDF, los cuales cargamos desde un directorio de Google Drive usando el framework Langchain.

```
from google.colab import drive
drive.mount('/content/drive')

dir="/content/drive/MyDrive/Colab Notebooks/TFG Alejandro/Rett"
```

```
from langchain.document_loaders import PyPDFLoader, DirectoryLoader
loader = DirectoryLoader(
    dir,
    glob="**/*.pdf",
    loader_cls=PyPDFLoader,
)

documents = loader.load()
```

Una vez hayamos escogido los documentos que queramos que use nuestra aplicación, debemos pasarlo a un formato que el modelo de lenguaje pueda incorporar.

```
from langchain.embeddings import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter

# Dividir los documentos en fragmentos más pequeños
text_splitter = CharacterTextSplitter(
    separator="\n",
    chunk_size=512,
    chunk_overlap=20,
    length_function=len
)
texts = []
for doc in documents:
    texts.extend(text_splitter.split_text(doc.page_content))

# Crear los embeddings utilizando OpenAI
embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")
```

En este código, se divide cada documento en chunks con un tamaño de 512 tokens y una solapación de 20 tokens, y se usa el modelo de embeddings “ada-002” para transformar cada token a su equivalente en un sistema de representación vectorial. En la siguiente figura se muestra como ejemplo uno de los vectores que conforman la base de datos, creada por medio de FAISS (Facebook AI Similarity Search).

```
from langchain.vectorstores import FAISS
db = FAISS.from_texts(texts, embeddings)

# Obtener los embeddings del índice FAISS
embeddings_array = db.index.reconstruct_n(0, db.index.ntotal)

# Convertir a lista para facilitar la lectura (opcional)
embeddings_list = embeddings_array.tolist()

# Mostrar el número de embeddings generados y el primero como
ejemplo
print(f"Número de embeddings generados: {db.index.ntotal}")
print(f"Primer embedding (dimensión {len(embeddings_list[0])}):
{embeddings_list[0]}")
```

```
Número de embeddings generados: 1353
Primer embedding (dimensión 1536): [-0.029281478375196457,
0.002531148726120591, 0.03168327733874321, -0.038990870118141174, -
0.007735574152320623, …
```

Falta ajustar un detalle de la configuración de la base de datos de vectores, que es el recuperador. En este caso, le decimos que realice búsquedas por similitud y recoja los 2 chunks con mayor relevancia.

```
retriever = db.as_retriever(search_type="similarity",
search_kwargs={"k": 2})
```

Con esto ya queda configurada la base de datos de vectores y podemos comenzar a usar la aplicación de forma práctica:

```
query="Qué puedes decir sobre el uso de trofenitide y cannabis en
pacientes de sindrome de Rett?"


docs = retriever.get_relevant_documents(query)

for i, doc in enumerate(docs):
    print(f"Documento {i + 1}:")
    print(doc.page_content)
    print("\n" + "="*80 + "\n")  # Separador para claridad

# Combina los documentos recuperados en un contexto único
context = "\n\n".join([doc.page_content for doc in docs])

# Construye el prompt para el modelo
prompt = f"""
Usa la siguiente información relevante para responder la pregunta:
{context}

Pregunta: {query}
```

```
Respuesta:
"""

from openai import OpenAI
import os

client = OpenAI(
    api_key=os.environ.get("OPENAI_API_KEY"),
)

response = client.chat.completions.create(
    model="gpt-3.5-turbo",  # Modelo a utilizar
    messages=[
        {"role": "user", "content": prompt}  # Mensaje del usuario
    ],
    temperature=0.3,          # Control de creatividad
    max_tokens=256,           # Máximo de tokens en la respuesta
    top_p=1,                  # Nucleus sampling
    frequency_penalty=0,      # Penalización por repetición de
palabras
    presence_penalty=0        # Penalización por introducir nuevos
temas
)

print(response.choices[0].message.content)
```

Aquí se declara la consulta, se crea una plantilla de prompt en la que se añade el contexto recuperado a la consulta, y se configura el modelo de lenguaje que se usará para generar la respuesta.

```
Documento 1:
o más. La trofinetida reduce la inflamación en el cerebro, aumenta la cantidad de una
proteína protectora en el cerebro y evita que algunas células lleguen a ser demasiado
activas.
Otros tratamientos se centran en el tratamiento de síntomas o comportamientos específicos
que se observan en el síndrome de Rett. Los medicamentos pueden ayudar con los problemas
del movimiento y la respiración, y con las crisis convulsivas.

================================================================================

Documento 2:
las estrategias de comunicación efectiva. El apoyo psicosocial a las familias es esencial.
Los enfoques farmacológicos tienen como objetivo mejorar los trastornos del sueño, los
problemas respiratorios, las crisis epilépticas, los movimientos estereotipados y el
bienestar general. Dado que los pacientes con RTT presentan un mayor riesgo de sufrir
arritmias potencialmente mortales asociadas a un intervalo QT prolongado, se deberá evitar
una serie de fármacos (como los antibióticos macrólidos).

================================================================================
```

Aquí se muestra de forma estructurada los 2 chunks que recoge para dar contexto al prompt.

La respuesta del modelo en este caso es la siguiente:

El uso de trofenitide en pacientes con síndrome de Rett parece ser beneficioso, ya que reduce la

inflamación en el cerebro, aumenta la cantidad de una proteína protectora y evita que algunas células se vuelvan demasiado activas. Por otro lado, el cannabis también puede ser útil en el tratamiento de algunos síntomas del síndrome de Rett, como problemas de movimiento, respiración y crisis convulsivas. Sin embargo, es importante consultar con un médico especialista antes de iniciar cualquier tratamiento con cannabis, ya que puede tener efectos secundarios y contraindicaciones en ciertos pacientes.

En el siguiente fragmento de código se llama al modelo sin RAG, es decir, sin contexto adicional en el prompt, para comparar la respuesta.

```
from openai import OpenAI

client = OpenAI(
    api_key=os.environ.get("OPENAI_API_KEY"),
)

response = client.chat.completions.create(
    model="gpt-3.5-turbo",  # Modelo a utilizar
    messages=[
        {"role": "user", "content": query}  # Mensaje del usuario
    ],
    temperature=0.3,           # Control de creatividad
    max_tokens=256,            # Máximo de tokens en la respuesta
    top_p=1,                   # Nucleus sampling
    frequency_penalty=0,       # Penalización por repetición de
palabras
    presence_penalty=0         # Penalización por introducir nuevos
temas
)

print(response.choices[0].message.content)
```

En este caso, la respuesta que genera el modelo es la siguiente:

No hay suficiente evidencia científica para respaldar el uso de trofenitide y cannabis en pacientes con síndrome de Rett. Siempre es importante consultar con un médico especializado antes de comenzar cualquier tratamiento, ya que cada paciente es único y puede responder de manera diferente a diferentes terapias. Es fundamental seguir las recomendaciones médicas y realizar un seguimiento adecuado para garantizar la seguridad y eficacia del tratamiento.

Como se puede apreciar, el modelo genérico da una respuesta poco relevante que no aporta demasiada información, mientras que el modelo es mucho más atrevido y es capaz de dar datos concretos de su uso.

17 Construcción de un RAG Biomédico.

17.1 Metodología

Aquí no pongo nada por ahora, será lo último

17.2 Selección de componentes del pipeline

Aquí explico cuales cuáles son los componentes que selecciono de los que se han presentado en los apartados anteriores. Justificó las decisiones que voy tomando.

Se explica qué embeddings se usan, qué parámetros de tamaño de chunks y del resto…

17.3 Selección del corpus de documentos

Aquí hablamos de PubMed y de por qué vamos a usar documentos de Síndrome de Rett. Hablamos de cómo seleccionamos los documentos (ensayos clínicos y documentos recientes)

Para poder recoger una gran cantidad de datos médicos relevantes y estudios actuales, hemos decidido usar PubMed como motor de búsqueda de artículos médicos.

Pubmed es una base de datos de libre acceso que permite buscar revistas médicas de la base de datos MEDLINE y otros repositorios. Debido a su eficiencia de búsqueda, su cobertura temática, su rigurosidad y constante actualización, resulta el motor de búsqueda más empleado por los expertos médicos.

En nuestro caso, hemos establecido un filtro de búsqueda con los siguientes parámetros. Queremos recoger artículos relacionados con el síndrome de rett (Rett syndrome), que hayan sido publicados en los últimos 5 años, que el artículo completo esté disponible de forma gratuita, y seleccionamos como tipo de artículo Informes de casos, Estudio clínico, Prueba clínica, Revisión y Revisión sistemática.

Lo ordenamos para tener primero los artículos más recientes, y recogemos los 20 primeros. Esto nos da como resultado el siguiente enlace:

https://pubmed.ncbi.nlm.nih.gov/?term=rett+syndrome&filter=datesearch.y_5&filter=simsearch2.ffrft&filter=pubt.casereports&filter=pubt.clinicalstudy&filter=pubt.clinicaltrial&filter=pubt.review&filter=pubt.systematicreview&sort=date

17.4 Uso de LLM Open Source. Mistral.

Aquí hablamos de que las pruebas se han hecho con modelos en la nube pero para el ejemplo de aplicación vamos a usar un modelo offline open source: explicamos en qué consiste y destacamos las ventajas que aporta

Enlazamos los lugares donde revisar la comparativa de rendimiento entre LLMs open source existentes, añadimos una imagen y justificamos por qué usar Mistral.

17.5 Construcción. Uso de Ollama.

En este apartado explicamos cómo con los componentes seleccionados del pipeline. Explico cómo voy añadiendo los documentos seleccionados en el apartado 3 a mi sistema.

También se explica qué es Ollama, para qué sire, cómo se instala y ejecuta.

17.6 Interfaz de usuario del sistema RAG. Chainlit.

Aquí hablaremos de cómo ofrecer el servicio a través de una web/http, de las distintas herramientas que hay, hablaremos de Chainlit y pondremos un poco de código explicando cómo funciona

17.7 Evaluación de resultados. RAGAS.

En esta parte final, si terminan bien las pruebas con RAGAS, se explica qué es, cómo funciona, cómo se crea un Golden standard sintético (estoy en ello) y se muestran los resultados obtenidos (con muchas imágenes)

18 Conclusiones y trabajos futuros

- Resumen y conclusiones de lo hecho
- Trabajos futuros
- Opinión personal

En este apartado de conclusiones, en el que además de aparecer un resumen de lo hecho y las conclusiones del trabajo, se deberá incluir cómo podría evolucionar y mejorarse el trabajo realizado, así como conclusiones desde un punto de vista personal (qué le ha supuesto la realización del TFG/TFM, qué nuevas herramientas ha utilizado o en qué herramientas ha profundizado, …).

19 Bibliografía

Memoria de verificación de originalidad*